



## 2.3 — Void functions (non-value returning functions)

👤 ALEX<sup>1</sup> 🕒 **OCTOBER 23, 2023**

In a prior lesson ([2.1 -- Introduction to functions](https://www.learncpp.com/cpp-tutorial/introduction-to-functions/)), we indicated that the syntax for a function definition looks like this:

```
returnType identifier() // identifier replaced with the name of your func
{
    // Your code here
}
```

Although we showed examples of functions that had return-type `void`, we did not discuss what this meant. In this lesson, we'll explore functions with a return type of `void`.

### Void return values

Functions are not required to return a value back to the caller. To tell the compiler that a function does not return a value, a return type of **void** is used. For example:

```
1  #include <iostream>
2
3  // void means the function does not return a value to the caller
4  void printHi()
5  {
6      std::cout << "Hi" << '\n';
7
8      // This function does not return a value so no return statement is
9      needed
10 }
11
12 int main()
13 {
14     printHi(); // okay: function printHi() is called, no value is returned
15
16     return 0;
17 }
```

In the above example, the `printHi` function has a useful behavior (it prints “Hi”) but it doesn't need to return anything back to the caller. Therefore, `printHi` is given a `void` return type.

When `main` calls `printHi`, the code in `printHi` executes, and “Hi” is printed. At the end of `printHi`, control returns to `main` and the program proceeds.

A function that does not return a value is called a **non-value returning function** (or a **void function**).

## Void functions don't need a return statement

A void function will automatically return to the caller at the end of the function. No return statement is required.

A return statement (with no return value) can be used in a void function -- such a statement will cause the function to return to the caller at the point where the return statement is executed. This is the same thing that happens at the end of the function anyway. Consequently, putting an empty return statement at the end of a void function is redundant:

```
1  #include <iostream>
2
3  // void means the function does not return a value to the caller
4  void printHi()
5  {
6      std::cout << "Hi" << '\n';
7
8      return; // tell compiler to return to the caller -- this is redundant
              // since the return will happen at the end of the function anyway!
9  } // function will return to caller here
10
11 int main()
12 {
13     printHi();
14
15     return 0;
16 }
```

### Best practice

Do not put a return statement at the end of a non-value returning function.

## Void functions can't be used in expression that require a value

Some types of expressions require values. For example:

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << 5; // ok: 5 is a literal value that we're sending to the
6      console to be printed
7      std::cout << ; // compile error: no value provided
8
9      return 0;
10 }
```

In the above program, the value to be printed needs to be provided on the right-side of the `std::cout <<`. If no value is provided, the compiler will produce a syntax error. Since the second call to `std::cout` does not provide a value to be printed, this causes an error.

Now consider the following program:

```
1 | #include <iostream>
2 |
3 | // void means the function does not return a value to the caller
4 | void printHi()
5 | {
6 |     std::cout << "Hi" << '\n';
7 | }
8 |
9 | int main()
10 | {
11 |     printHi(); // okay: function printHi() is called, no value is returned
12 |
13 |     std::cout << printHi(); // compile error
14 |
15 |     return 0;
16 | }
```

The first call to `printHi()` is called in a context that does not require a value. Since the function doesn't return a value, this is fine.

The second function call to function `printHi()` won't even compile. Function `printHi` has a `void` return type, meaning it doesn't return a value. However, this statement is trying to send the return value of `printHi` to `std::cout` to be printed. `std::cout` doesn't know how to handle this (what value would it output?). Consequently, the compiler will flag this as an error. You'll need to comment out this line of code in order to make your code compile.

## Tip

Some statements require values to be provided, and others don't.

When we have a statement that consists of just a function call (e.g. the first `printHi()` in the above example), we're calling a function for its behavior, not its return value. In this case, we can call either a non-value returning function, or we can call a value-returning function and just ignore the return value.

When we call a function in a context that requires a value (e.g. `std::cout`), a value must be provided. In such a context, we can only call value-returning functions.

```
1 | #include <iostream>
2 |
3 | // Function that does not return a value
4 | void returnNothing()
5 | {
6 | }
7 |
8 | // Function that returns a value
9 | int returnFive()
10 | {
11 |     return 5;
12 | }
13 |
14 | int main()
15 | {
16 |     // When calling a function by itself, no value is required
17 |     returnNothing(); // ok: we can call a function that does not return
18 | a value
19 |     returnFive();    // ok: we can call a function that returns a value,
20 | and ignore that return value
21 |
22 |     // When calling a function in a context that requires a value (like
23 | std::cout)
24 |     std::cout << returnFive();    // ok: we can call a function that
25 | returns a value, and the value will be used
26 |     std::cout << returnNothing(); // compile error: we can't call a
27 | function that returns void in this context
28 |
29 |     return 0;
30 | }
```

## Returning a value from a void function is a compile error

Trying to return a value from a non-value returning function will result in a compilation error:

```
1 | void printHi() // This function is non-value returning
2 | {
3 |     std::cout << "In printHi()" << '\n';
4 |
5 |     return 5; // compile error: we're trying to return a value
6 | }
```

## Quiz time

### Question #1

Inspect the following programs and state what they output, or whether they will not compile.

1a)

```
1 | #include <iostream>
2 |
3 | void printA()
4 | {
5 |     std::cout << "A\n";
6 | }
7 |
8 | void printB()
9 | {
10 |    std::cout << "B\n";
11 | }
12 |
13 | int main()
14 | {
15 |     printA();
16 |     printB();
17 |
18 |     return 0;
19 | }
```

[Hide Solution](#) (javascript:void(0))<sup>3</sup>

| This program prints the letters A and B on separate lines.

1b)

```
1 | #include <iostream>
2 |
3 | void printA()
4 | {
5 |     std::cout << "A\n";
6 | }
7 |
8 | int main()
9 | {
10 |    std::cout << printA() << '\n';
11 |
12 |    return 0;
13 | }
```

[Hide Solution](#) (javascript:void(0))<sup>3</sup>

| This program does not compile. Function `printA()` returns `void`, which can't be sent to `std::cout` to be printed. This will produce a compile error.



## [Next lesson](#)

2.4 [Introduction to function parameters and arguments](#)

4



## [Back to table of contents](#)

5



## [Previous lesson](#)

2.2 [Function return values \(value-returning functions\)](#)


6

7



**B** **U** **URL** **INLINE CODE** **C++ CODE BLOCK** **HELP!**

Leave a comment...

 Name\*

@ Email\* | ?

Notify me about replies:



POST COMMENT

🔍 Find a mistake? Leave a comment above!?

👤 Avatars from <https://gravatar.com/><sup>10</sup> are connected to your provided email address.

50 COMMENTS

Newest ▼

**Call Any Vegetable**

🕒 February 27, 2024 3:40 pm

If I understand this correctly, `std::cout` is in essence a function call...thanks to the use of the operator=.

👍 0

➡ Reply

**Call Any Vegetable**🗨 Reply to [Call Any Vegetable](#)<sup>11</sup> 🕒 February 27, 2024 3:47 pm

Brain fade. I meant the operator<<.

Come to think of it, would `cin` also be a function call? It's asking for a value to be input, not output. I just want to get the terminology correct.

👍 0

➡ Reply

**Alex** Author🗨 Reply to [Call Any Vegetable](#)<sup>12</sup> 🕒 February 29, 2024 9:05 pm

No. `operator<<` is essentially a function call, and `std::cin` (and the object being input to) is an object (of type `std::istream`).

👍 3

➡ Reply

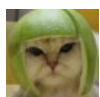
**IceFloe**

🕒 February 19, 2024 4:25 pm

so the other types of the function always return a value? And one more thing, please explain again step by step how the function is called and the value is returned back to the calling party, because void also returns a value, it's just that most likely the calling party does not store its value for further use.

👍 0

➡ Reply

**Alex** Author🗨 Reply to [IceFloe](#)<sup>13</sup> 🕒 February 19, 2024 9:22 pm

Non-void functions return a value. Void functions do not return a value.

We talk about more about value passing and returning in the next lesson.

👍 0

➡ Reply

**JPerk**Reply to [Alex](#)<sup>14</sup> ⌚ March 9, 2024 12:36 am

we need more Alex's in our lives.  
(thx author)

✎ Last edited 1 month ago by JPerk

👍 1 ➡ Reply

**nava**

⌚ February 19, 2024 4:03 am

best learning site, even compared to "compact" and "learning" apps like sololearn. It's way better

👍 6 ➡ Reply

**Ilumi**

⌚ February 16, 2024 6:21 am

nice

👍 0 ➡ Reply

**AbeerOrTwo**

⌚ December 15, 2023 10:07 am

day tresss

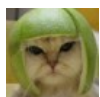
👍 1 ➡ Reply

**Abdullah**

⌚ November 26, 2023 3:01 pm

Anyone help clarify what this sentence is trying to say. (It's located in the Tip section, 2nd paragraph) "In this case, we can call either a non-value returning function, or we can call a value-returning function and just ignore the return value."

👍 0 ➡ Reply

**Alex** AuthorReply to [Abdullah](#)<sup>15</sup> ⌚ November 28, 2023 11:53 am

In addition to logan's answer, this is illustrated in the subsequent example:



```
1 // When calling a function by itself, no value is required
2 returnNothing(); // ok: we can call a function that does not return
3 a value
  returnFive();    // ok: we can call a function that returns a
                    // value, and ignore that return value
```

 Last edited 4 months ago by Alex

 2  Reply



**logan**

 Reply to [Abdullah](#) <sup>15</sup>  November 27, 2023 4:25 pm

yeah, so basically what it is saying is if we are only looking for the behavior of a function and not just what it is returning, then we can basically switch between using a void function and as an example, an int function for a singular call with no value needing to be actually used.

Say the void function "printHi()" is called with "printHi();" under the main function. That function would print "Hi." to the console, and not return a value within its function because it is a void function only meant to print the word "Hi." when called.

Now, say you had an int function such as "numberOfHellos()" that returns a value of 3 because it counts the number of hello's that are said to you as you walk into a room, but also prints "Hi." as a secondary feature that isn't too tied into returning the 3 hello's. You could call this function under the main function with "numberOfHellos();" and it would give you the same output as the function "printHi()", but it contains the return value "3" which is ignored.

This is just what I believe is meant by this, but of course this example was completely arbitrary and not actually taking into account what other things might be recommended for creating functions.

 2  Reply

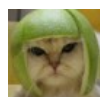


**Abdullah**

 Reply to [logan](#) <sup>16</sup>  November 29, 2023 10:32 am

Thanks for the clarification. So may I ask, why use a void function if you can use a returning-function that can do both things: not return values and return values?

 0  Reply



**Alex** Author

 Reply to [Abdullah](#) <sup>17</sup>  December 1, 2023 12:20 pm

Because functions that don't need to return a value shouldn't.

For example, the `printHi()` function in this lesson -- what value would it return?

👍 2    ➡ Reply



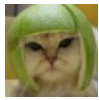
**jhdthdkfgjkhdfg**

🕒 October 27, 2023 1:45 am

"A function that does not return a value is called a non-value returning function (or a void function)."

It is mostly called "a procedure" I think.

👍 0    ➡ Reply



**Alex** Author

🗨️ Reply to [jhdthdkfgjkhdfg](#)<sup>18</sup> 🕒 October 27, 2023 11:12 am

The term "procedure" typically isn't used in C++.

👍 2    ➡ Reply



**Egor**

🕒 September 2, 2023 1:48 pm

I was wondering what kind of console output is more preferable and correct:

```
void returnFive() {
std::cout << 5 << '\n';
}

int return5() {
return 5;
}

int main() {
returnFive();
std::cout << return5() << '\n';

return 0;
}
```

I have seen the int variant way more often but i'm still curious about the difference, especially when there are variables instead of literals

🔗 Last edited 7 months ago by Egor

0

Reply

**park** Reply to [Egor](#)<sup>19</sup> October 18, 2023 6:45 pm

I think the name of the function should be "printFive", not "returnFive", because it **prints** out "5" on the console window, and does not **return** any value. Then, both would look equally preferable to me.

2

Reply

**Will** Reply to [Egor](#)<sup>19</sup> September 3, 2023 3:09 pm

Both will work, but one is definitely better than the other.

In general, you should separate the behind-the-scenes stuff from the stuff the user will see, so you can modify one without affecting the other.

So it would be best practice to do

```
1 | std::cout<< return5() << '\n';
```

Replace return5() with a function that takes in a variable and does tasks with it, and you can see why tying the user output with the actual functionality in a piece of code can become troublesome.

3

Reply

**Krishnakumar**

August 23, 2023 3:36 pm

>Do not put a return statement at the end of a non-value returning function.

However, dependent on some logic incorporated in the function, a program can conditionally return early if the return statement is placed earlier.

e.g.

```
1 | void printVal(int x) {  
2 |     if (x < 0) {  
3 |         return;  
4 |     }  
5 |     std::cout << "The number is: " << x << '\n';  
6 | }
```

I know we haven't covered `if` statements and conditionals yet, but a `return` statement placed before the end of the function body could generally be useful in many other situations in functions with a void return type.

👍 0    ➡ Reply



**Alex**    Author

➡ Reply to [Krishnakumar](#)<sup>20</sup>    ⌚ August 28, 2023 12:40 pm

Yes, but such return statements are not placed at the *end* of a non-value returning function.

👍 7    ➡ Reply



**Krishnakumar**

➡ Reply to [Alex](#)<sup>21</sup>    ⌚ October 9, 2023 12:48 pm

Ah. Apologies for the bother, and thanks for the clarification, Alex.

👍 2    ➡ Reply



**Foo**

⌚ July 14, 2023 6:19 am

hi alex, why is it that when a function with the void type is called in the `std::cout` operator, it leads to an error, but when a function with the `int` type is called in the same way, there is no error and the program works as expected? this confused me a bit, because it seems like the functions do the same thing (print some text), just one returns a value, and the other does not.

and it's also unclear, we can call a function not only by simply writing its name, parentheses and semicolons, but also through the `std::cout` operator?

✎ Last edited 9 months ago by Foo

👍 3    ➡ Reply




**Alex**    Author

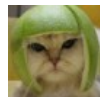
➡ Reply to [Foo](#)<sup>22</sup>    ⌚ July 14, 2023 1:03 pm

`std::cout` requires value to output as an operand. If the function returns a value, that value can be used as the operand. If the function does not return a value then there is no value to use as an operand, and the compiler is making you aware of the fact that you cannot print void to the console.

If you are calling a function in a context that does not require a value (e.g. as a stand-alone function call) then it doesn't matter whether the function returns a value or not.

 5 Reply**Foo** Reply to [Alex](#)<sup>23</sup>  July 15, 2023 11:20 am

I figured out the void and int types, but I still don't understand this line "std::cout << print();" how can I print a function call on the screen? it prints the text by itself (in this case, the letter A), so we also put it in the std::cout operator???

 0 Reply**Alex**

Author

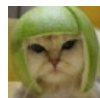
 Reply to [Foo](#)<sup>24</sup>  July 16, 2023 2:35 am

`std::cout << printA();` doesn't print a function call on the screen. If `printA()` had a non-void return type, `printA()` would first be called (which would do whatever the function is defined to do), and then the return value of the function would be printed on the screen.

But because `printA()` does not return a value (it has a void return type), trying to print its return value to the screen is an error.

 0 Reply**Foo** Reply to [Alex](#)<sup>23</sup>  July 15, 2023 10:31 am

could you explain in more detail the last paragraph of your answer and what does stand-alone function call mean? this confused me even more

 0 Reply**Alex**

Author

 Reply to [Foo](#)<sup>25</sup>  July 16, 2023 2:17 am

By stand-alone function call, I mean an expression statement that consists of just a function call. e.g.

```
1 | print(); // if print returns a value, it will be
   | discarded since it is not used
```

 0 Reply**Zuma** June 17, 2023 6:26 am

sorry maybe i'm dumb, but in the last example, why did it print the 'A'? even when you didn't call it. it should only print the return. right?

👍 0    ➡ Reply



**Wizard**

🗨 Reply to [Zuma](#) ⌚ June 18, 2023 9:27 pm

firs executed in print function `std::cout << "A";` and `return 5` then in main function executed `std::cout << print();`.

So, A print first and then the output of print function is printed as 5

👍 5    ➡ Reply



**Tresco**

⌚ May 26, 2023 5:08 am



A void function somehow "assumes" that there are no errors in the function. I understood that it has nothing to return, it doesn't have to return anything, but if there are errors? Doesn't it return an error like `int main()`?

Conceptual as `main()`:

`void funcName() => (EXIT_SUCCESS/void, EXIT_FAILURE/error_code)`

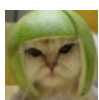
- all good => void
- some errors => error\_code

Of course, `main()` returns a 0 or a number because of the "int" return type, but still. Am I to assume that beyond `main()`, error handling is the programmer's problem? How do you return an error from a void, no-return function? Not something specific ... general like `main()`, hey dumb you got an error!

The only logical answer to me is "don't use the void function if you have something to return, even errors", right? So... what's the point of the void function if you can't "manipulate" it properly?

Maybe I just don't understand concepts and I'm sorry for that.

👍 0    ➡ Reply



**Alex** Author

🗨 Reply to [Tresco](#) ⌚ May 29, 2023 11:24 pm

There are many different ways to indicate to the caller that an error has occurred -- using a return value is one possible option, but there are others. For example, out parameters and exceptions are two other ways to indicate that an error has occurred that doesn't use a return value.

We discuss this topic a bit more in lesson <https://www.learncpp.com/cpp-tutorial/detecting-and-handling-errors/>

👍 3    ➡ Reply



**Tresco**

🗨 Reply to [Tresco](#) ⌚ May 29, 2023 4:37 am

PS

Some explanation are here:

<https://stackoverflow.com/questions/40582977/why-use-void-with-a-function>

<https://stackoverflow.com/questions/11560068/if-void-does-not-return-a-value-why-do-we-use-it>

Bottom line conclusion, for me at least are:

- by design a function need to have a return type;
- void is in fact a data type, but is an incomplete defined data type;
- void tell to compiler to not allocate resources to store that value.

We can use a function like:

```
int myVoidFunctionPrint()
{
    std::cout << "A\n";

    return 0;
}
```

for console output and not use the return value.

This solution is very confusing, it doesn't explain exactly the "scope" of the returned value. Moreover, resources are allocated for the returned value. For thousands of such functions, this can become a problem (program optimization).

Using the void concept has several purposes in C++, but they are a bit more complicated at my level, so I skip them, for now.

About errors:

The main() is the entry/exit point of our program (application). The operating system need to know the state of application (not running, is running, is terminated, ...). Are like 404, 200, ... code for a website. When the main() return 0, meaning "the program ran, zero errors appear". When the main() return 1, meaning "the program ran, at least one error appear", is not important how many.

So main() returns 0 or 1 for the purpose of the operating system, not the user.

The system does not care about every function in your application, only about the state of the application, EXIT\_SUCCESS or EXIT\_FAILURE, to know what to do next. It's the programmer's problem what he does in the application.

👍 0    ➡ Reply

**cPlusPlus\_Warlord**

🕒 April 28, 2023 12:47 pm

Is there a way to pass a variable number of arguments/types to a function in C++? Python has this capability with its `"*args"` syntax. For example in python this can be achieved by the below:

```
1 def func(*args):  
2     for i in args:  
3         return i
```

The above functionality enables you to pass any amount of arguments and data types (noting Python is a dynamically typed language).

👍 0    ➡ Reply

**Alex**    Author

➡ Reply to cPlusPlus\_Warlord    🕒 May 2, 2023 1:57 pm

Yes. See <https://www.learncpp.com/cpp-tutorial/ellipsis-and-why-to-avoid-them/>

👍 2    ➡ Reply

**Tayyaba**

🕒 February 15, 2023 8:23 pm

Can we use arguments in void return type

As

Void Shop(int a,int b)

??

👍 0    ➡ Reply

**Alex**    Author

➡ Reply to Tayyaba    🕒 February 16, 2023 7:52 pm

Yes, absolutely. There is an example of this in the next lesson: `void printValue(int x)`

👍 2    ➡ Reply

**Pedro**

🕒 October 24, 2022 12:48 pm

Hello there!

I can't seem to compile anything with an **early return**.



Example:

```
1 | #include <iostream>
2 |
3 | int print()
4 | {
5 |     std::cout << "A";
6 |     return 5;
7 |     std::cout << "B";
8 |
9 | }
10 |
11 | int main()
12 | {
13 |     std::cout << print();
14 |     return 0;
15 | }
```

I feel it should be possible to compile this (using Visual 2022) as you've mentioned above, but for some reason it doesn't work. Thanks for the feedback!

👍 1    ➡ Reply



**Kedja**

👤 Reply to [Pedro](#) 🕒 October 25, 2022 7:53 am

it's probably because you have the setting to treat warnings as errors and warnings set to level 4 as we did in the beginning of the series.

Go back and disable that if you want this to compile.

👍 3    ➡ Reply



**Pedro**

👤 Reply to [Kedja](#) 🕒 October 26, 2022 4:09 pm

Yup. That did the trick.

Thanks for the reply (I can sleep again)!

👍 3    ➡ Reply

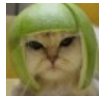


**DUCKONAROLL**

🕒 August 9, 2022 4:29 pm

Why were early returns formerly frowned upon?

👍 2    ➡ Reply

**Alex** AuthorReply to [DUCKONAROLL](#) ⌚ August 10, 2022 2:36 pm

There are some developers who believe that having one entry point and one exit point makes a function easier to understand. Early returns were also historically problematic when some kind of cleanup is required (E.g. if at the end of the function you need to close a file, or deallocate some memory). Modern C++ has ways to deal with most of the cleanup issues now (e.g. smart pointers).

👍 7 ➡ Reply

**Krishnakumar**Reply to [Alex](#) ⌚ October 9, 2023 12:49 pm

Looks like wording about early returns were removed from this lesson at some point?

👍 0 ➡ Reply

**Alex** AuthorReply to [Krishnakumar](#) ⌚ October 10, 2023 9:14 pm

Yep, it was moved to <https://www.learncpp.com/cpp-tutorial/introduction-to-if-statements/> where I had the concepts available to show an example that didn't result in unreachable code.

👍 0 ➡ Reply

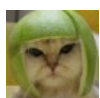
**Krishnakumar**

⌚ June 15, 2022 2:46 am

If no value is provided, the **program will error**. Since the second call to `std::cout` does not provide a value to be printed, the **compiler will error**.

Somehow, feels odd when reading this. Program will error, compiler will error? We understand what is meant, but new learners (especially non-native speakers) can probably benefit from a simpler rephrasing.

👍 3 ➡ Reply

**Alex** AuthorReply to [Krishnakumar](#) ⌚ June 15, 2022 10:56 am

Amended. Thanks!

👍 6 ➡ Reply

**Krishnakumar**Reply to [Alex](#) May 22, 2023 3:49 pm

Thank you!

👍 0

➡ Reply

**阮文昊**

April 17, 2022 6:17 am

```
1 | using namespace std; //why don't you use it???
```

👍 1

➡ Reply

**Amir**Reply to [阮文昊](#) July 28, 2022 1:01 pm

The reason behind this that in C++ we need to use STD for referring to the standard library which has it's own set of functionality. simply by using Namespace std you ignore the fundamental of knowing which functions belong where. it's an early game convenience though the later you get into development it will be a good indicator of the Standard library. it will avoid lots of conflict.

👍 2

➡ Reply

**Landen**Reply to [阮文昊](#) April 17, 2022 9:46 pm

You could run into conflicts. For example, if you wanted to swap the values of two variables, you can create a **swap** function. Let's say I want to make **swap** work with more than one type (I can use it to swap 2 ints, later on I can use it to swap 2 chars, etc). The way this templated function works doesn't need to be understood, just that this will not compile (continue reading below the program).

```

1  #include <iostream>
2  using namespace std;
3
4  // Function to swap 2 variables of the same type
5  template<typename T>
6  void swap(T &x, T &y)
7  {
8      T temp = x;
9      x = y;
10     y = temp;
11 }
12
13 int main()
14 {
15     int x = 5;
16     int y = 10;
17
18     swap(x, y);
19
20     cout << "Value of x: " << x << endl; // Should be 10
21     cout << "Value of y: " << y << endl; // Should be 5
22
23     return 0;
24 }

```

However a **swap** function that I just made already exists in the **standard namespace**. So you will get a compiler error stating the function call is ambiguous. Should the compiler use the function I made, or should it use the function of the **standard namespace**? It doesn't explicitly know. Here is the error:

```

1  g++ /tmp/6QLFM0tVem.cpp
2  /tmp/6QLFM0tVem.cpp: In function 'int main()':
3  /tmp/6QLFM0tVem.cpp:15:14: error: call of overloaded 'swap(int&,
4  int&)' is ambiguous // ERROR DESCRIPTION
5      15 |         swap(x, y);
6          |         ^
7  /tmp/6QLFM0tVem.cpp:6:6: note: candidate: 'void swap(T&, T&) [with
8  T = int]' // STANDARD NAMESPACE FUNCTION
9      6 | void swap(T &x, T &y) // MY FUNCTION
10         |         ^~~~

```

Taking away **using namespace std** and correcting **cout** and **endl** to **std::cout** and **std::endl** will allow you to compile the program and see the proper output.

👍 38    ➡ Reply

## Links

1. <https://www.learncpp.com/author/Alex/>
2. <https://www.learncpp.com/cpp-tutorial/introduction-to-functions/>
3. [javascript:void\(0\)](javascript:void(0))
4. <https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/>
5. <https://www.learncpp.com/>
6. <https://www.learncpp.com/cpp-tutorial/function-return-values-value-returning-functions/>
7. <https://www.learncpp.com/void-functions-non-value-returning-functions/>
8. <https://www.learncpp.com/cpp-tutorial/chapter-12-summary-and-quiz/>
9. <https://www.learncpp.com/cpp-tutorial/class-templates/>
10. <https://gravatar.com/>
11. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-594097>
12. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-594098>
13. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-593828>
14. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-593844>
15. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-590261>
16. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-590305>
17. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-590362>
18. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-589158>
19. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-586490>
20. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-585977>
21. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-586272>
22. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-583786>
23. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-583804>
24. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-583843>
25. <https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/#comment-583839>